CHAPTER

7 Authentication and Authorization

One of the most common ways to control access to computer systems is to identify who is at the keyboard (and prove that identity), and then decide what they are allowed to do. These twin controls, *authentication* and *authorization*, respectively, ensure that authorized users get access to the appropriate computing resources, while blocking access to unauthorized users. Authentication is the means of verifying who a person (or process) is, while authorization determines what they're allowed to do. This should always be done in accordance with the principle of least privilege—giving each person only the amount of access they require to be effective in their job function, and no more.

Authentication

Authentication is the process by which people prove they are who they say they are. It's composed of two parts: a public statement of identity (usually in the form of a *username*) combined with a private response to a challenge (such as a *password*). The secret response to the authentication challenge can be based on one or more factors—something you know (a secret word, number, or passphrase for example), something you have (such as a smartcard, ID tag, or code generator), or something you are (like a biometric factor like a fingerprint or retinal print). A password by itself, which is a means of identifying yourself through something only you should know (and today's most common form of challenge response), is an example of *single-factor authentication*. This is not considered to be a strong authentication method, because a password can be intercepted or stolen in a variety of ways—for example, passwords are frequently written down or shared with others, they can be captured from the system or the network, and they are often weak and easy to guess.

Imagine if you could only identify your friends by being handed a previously agreed secret phrase on a piece of paper instead of by looking at them or hearing their voice. How reliable would that be? This type of identification is often portrayed in spy movies, where a secret agent uses a password to impersonate someone the victim is supposed to meet but has never seen. This trick works precisely because it is so fallible—the password is the only means of identifying the individual. Passwords are just not a good way of authenticating someone.

Unfortunately, password-based authentication was the easiest type to implement in the early days of computing, and the model has persisted to this day.

Other single-factor authentication methods are better than passwords. Tokens and smart cards are better than passwords because they must be in the physical possession of the user. Biometrics, which use a sensor or scanner to identify unique features of individual body parts, are better than passwords because they can't be shared—the user must be present to log in. However, there are ways to defeat these methods. Tokens and cards can be lost or stolen, and biometrics can be spoofed. Yet, it's much more difficult to do that than to steal or obtain a password. Passwords are the worst possible method of proving an identity, despite being the most common method.

Multifactor authentication refers to using two or more methods of checking identity. These methods include (listed in increasing order of strength):

- Something you know (a password or PIN code)
- Something you have (such as a card or token)
- Something you are (a unique physical characteristic)

Two-factor authentication is the most common form of multifactor authentication, such as a password-generating token device with an LCD screen that displays a number (either time based or sequential) along with a password, or a smart card along with a password. Again, passwords aren't very good choices for a second factor, but they are ingrained into our technology and collective consciousness, they are built into all computer systems, and they are convenient and cheap to implement. A token or smart card along with biometrics would be much better—this combination is practically impossible to defeat. However, most organizations aren't equipped with biometric devices.

The following sections provide a more detailed introduction to these types of authentication systems available today:

- Systems that use username and password combinations, including Kerberos
- Systems that use certificates or tokens
- Biometrics

Usernames and Passwords

In the familiar method of password authentication, a challenge is issued by a computer, and the party wishing to be identified provides a response. If the response can be validated, the user is said to be authenticated, and the user is allowed to access the system. Otherwise, the user is prevented from accessing the system.

Other password-based systems, including Kerberos, are more complex, but they all rely on a simple fallacy: they trust that anyone who knows a particular user's password is that user. Many password authentication systems exist. The following types of systems are commonly used today:

- Local storage and comparison
- Central storage and comparison

- Challenge and response
- Kerberos
- One-time password (OTP)

Each type of system is discussed in turn next.

Local Storage and Comparison

Early computer systems did not require passwords. Whoever physically possessed the system could use it. As systems developed, a requirement to restrict access to the privileged few was recognized, and a system of user identification was developed. User passwords were entered in simple machine-resident databases by administrators and were provided to users.

Often, passwords were stored in the database in plaintext format (unencrypted), because protecting them wasn't really a high priority. Anyone who was able to open and read the file could determine what anyone else's password was. The security of the database relied on controlling access to the file, and on the good intentions of all the administrators and users. Administrators were in charge of changing passwords, communicating changes to the users, and recovering passwords for users who couldn't remember them. Later, the ability for users to change their own passwords was added, as was the ability to force users to do so periodically. Since the password database contained all information in plaintext, the algorithm for authentication was simple—the password was entered at the console and was simply compared to the one in the file.

This simple authentication process was, and still is, used extensively for off-the-shelf and custom applications that require their own internal authentication processes. They create and manage their own stored-password file and do no encryption. Security relies on the protection of the password file. Because passwords can be intercepted by rogue software, these systems are not well protected.

Securing Passwords with Encryption and Securing the Password File In time, a growing recognition of the accessibility of the password file, along with some high-profile abuses, resulted in attempts to hide the password file or strengthen its defense. In early Unix systems, passwords were stored in a file called /etc/passwd. This file was world-readable (meaning that it could be opened and read by all users) and contained all the passwords in encrypted form. Nevertheless, the encryption was weak and easy to defeat. Blanking the password field (possible after booting from a CD) allowed a user to log in with no password at all. Again, after several highly publicized compromises, the system was redesigned to what it is today. In most modern Unix systems, the usernames are stored in the /etc/passwd file but the passwords are stored in a separate file, known as a shadow password file and located in /etc/shadow. It contains the encrypted passwords and is not world-readable. Access is restricted to system administrators, thus making an attack from a regular user account more difficult.

Much like early Unix systems, early versions of Windows used easily crackable password (.pwd) files. Similarly, Windows NT passwords were saved in the Security Account Manager (SAM), which could be modified to change the passwords contained in it, or subjected to brute-force attacks to obtain the passwords. Later versions of Windows added the syskey utility, which added a layer of protection to the database in the form of additional encryption.

However, attack tools were created that could be used to extract the password hashes from syskey-protected files.

Numerous freely available products can crack Windows and Unix passwords. Two of the most famous are LC4 (formerly known as LOphtCrack) and John the Ripper. These products typically work by using a combination of attacks: a dictionary attack (using the same algorithm as the operating system to hash words in a dictionary and then compare the result to the password hashes in the password file), heuristics (looking at the things people commonly do, such as create passwords with numbers at the end and capital letters at the beginning), and brute force (checking every possible character combination).

Another blow to Windows systems that are protected by passwords is the availability of a bootable Linux application that can replace the Administrator's password on a stand-alone server. If an attacker has physical access to the computer, they can take it over—though this is also true of other operating systems using different attacks.

Protection for account database files on many operating systems was originally very weak, and may still be less than it could be. Administrators can improve security by implementing stronger authorization controls (file permissions) on the database files.

In any case, ample tools are available to eventually compromise passwords if the machine is in the physical possession of the attacker, or if the attacker can obtain physical possession of the password database. That's why every system should be physically protected. Centralized account databases and authentication systems should be protected with extra precautions. In addition, user training and account controls can strengthen passwords and make the attacker's job harder—perhaps hard enough that the attacker will move on to easier pickings.

Today, many off-the-shelf applications now use the central authentication system already in use by the organization, such as Lightweight Directory Access Protocol (LDAP) or Active Directory. They rely on the existing credentials of the user, instead of maintaining their own password databases. This is much better from a security standpoint, as well as easier for the end users. Single sign-on (SSO) allows users to authenticate to applications using their current credentials, without being challenged, which improves the user experience.

Central Storage and Comparison

When passwords are encrypted, authentication processes change. Instead of doing a simple comparison, the system must first take the user-entered, plaintext password and encrypt it using the same algorithm used for its storage in the password file. Next, the newly encrypted password is compared to the stored encrypted password. If they match, the user is authenticated. This is how many operating systems and applications work today.

How does this change when applications are located on servers that client workstations must interface with? What happens when centralized account databases reside on remote hosts? Sometimes the password entered by the user is encrypted, passed over the network in this state, and then compared by the remote server to its stored encrypted password. This is the ideal situation. Unfortunately, some network applications transmit passwords in cleartext—telnet, FTP, rlogin, and many others do so by default. Even systems with secure local, or even centralized, network logon systems may use these and other applications which then transmit passwords in cleartext. If attackers can capture this data in flight, they can use it to log in as that user. In addition to these network applications, early remote authentication algorithms (used to log in via dial-up connections), such as Password Authentication Protocol (PAP), also transmit cleartext passwords from client to server.

What's in a Hash?

A hash function is a mathematical formula that converts a string of characters (text or numbers) to a numeric code (commonly called a hash). These functions are very important to encryption methods, and thus to authentication systems that require something (like a password) to be hidden. They've been around since the 1970s.

In theory, a hash is one-way code, which means you can create it, but not reverse it. It's like being able to encrypt something without being able to decrypt it. How is this useful? When one computer creates a hash, another computer can use exactly the same inputs to create another hash, and compare the two. If they match, the inputs are the same.

In practice, modern hashes can be cracked using advanced techniques. But they are still some of the most useful tools we have for obfuscating data.

Secure Hash Algorithm version 1 (SHA-1) and Message Digest version 5 (MD5) are the most widely known modern hash functions.

CHAP and MS-CHAP

One solution to the problem of securing authentication credentials across the network so they are not easily intercepted and replayed is to use the challenge and response authentication algorithms Challenge Handshake Authentication Protocol (CHAP, described in RFC 1994) and the Microsoft version, MS-CHAP (RFC 2433). These protocols use the Message Digest version 5 (MD5). The server that receives the request for access issues a challenge code, and the requestor responds with an MD5 hash of the code and password. The server then compares that hash to its own hash made from the same code and password. If they are the same, the user is authenticated.

In addition to more secure storage of credentials, version 2 of MS-CHAP (MSCHAPv2, described in RFC 2759) requires mutual authentication—the user must authenticate to the server, and the server must also prove its identity. To do so, the server encrypts a challenge sent by the client. Since the server uses the client's password to do so, and only a server that holds the account database in which the client has a password could do so, the client is also assured that it is talking to a valid remote access server. This is a stronger algorithm, although it is not unbreakable. MSCHAPv2 has been found to be vulnerable to brute force attacks that can be performed within a timeframe of minutes to hours on modern hardware.

Kerberos

Kerberos is a network authentication system based on the use of *tickets*. In the Kerberos standard (RFC 1510), passwords are key to the system, but in some systems certificates may be used instead. Kerberos is a complex protocol developed at the Massachusetts Institute of Technology to provide authentication in a hostile network. Its developers, unlike those of some other network authentication systems, assumed that malicious individuals as well as curious users would have access to the network. For this reason, Kerberos has designed into it various facilities that attempt to deal with common attacks

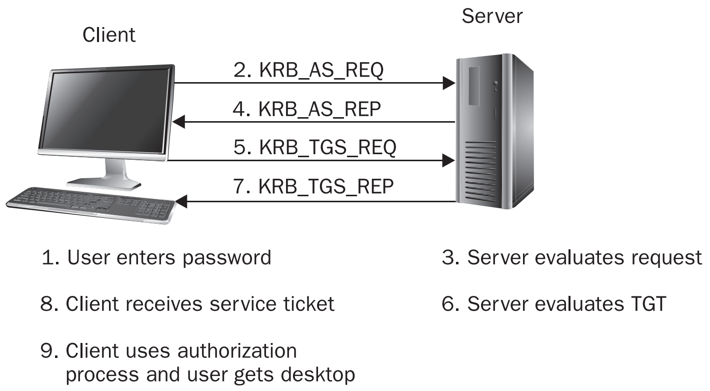
Client
Server
2. KRB_AS_REQ
4. KRB_AS_REP
5. KRB_TGS_REQ
7. KRB_TGS_REP
1. User enters password
3. Server evaluates request
8. Client receives service ticket
6. Server evaluates TGT
9. Client uses authorization process and user gets desktop

Figure 7-1 The Kerberos authentication system uses tickets and a multistep process.

on authentication systems. The Kerberos authentication process follows these steps, which are illustrated in Figure 7-1:

1. A user enters their password.
2. Data about the client and possibly an *authenticator* is sent to the server. The authenticator is the result of using the password (which may be hashed or otherwise manipulated) to encrypt a timestamp (the clock time on the client computer). This authenticator and a plaintext copy of the timestamp accompany a login request, which is sent to the Kerberos authentication server (AS)—this is the KRB_AS_REQ message. This is known as pre-authentication and may not be part of all Kerberos implementations.

NOTE Typically, both the AS and the Ticket Granting Service (TGS) are part of the same server, as is the Key Distribution Center (KDC). The KDC is a centralized database of user account information, including passwords. Each Kerberos realm maintains at least one KDC (a realm being a logical collection of servers and clients, comparable to a Windows domain).

3. The KDC checks the timestamp from the workstation against its own time. The difference must be no more than the authorized time skew (which is five minutes, by default). If the time difference is greater, the request is rejected.
4. The KDC, since it maintains a copy of the user's password, can use the password to encrypt the plaintext copy of the timestamp and compare the result to the authenticator. If the results match, the user is authenticated, and a ticket-granting ticket (TGT) is returned to the client—this is the KRB_AS_REP message.
5. The client sends the TGT to the KDC with a request for the use of a specific resource, and it includes a fresh authenticator. The request might be for resources local to the client computer or for network resources. This is the KRB_TGS_REQ message, and it is handled by the TGS.

6. The KDC validates the authenticator and examines the TGT. Since it originally signed the TGT by encrypting a portion of the TGT using its own credentials, it can verify that the TGT is one of its own. Since a valid authenticator is present, the TGT is also less likely to be a replay. (A captured request would most likely have an invalid timestamp by the time it is used—one that differs by more than the skew time from the KDC's clock.)
7. If all is well, the KDC issues a service ticket for the requested resource—this is the KRB_TGS_REP message. Part of the ticket is encrypted using the credentials of the service (perhaps using the password for the computer account on which the service lies), and part of the ticket is encrypted with the credentials of the client.
8. The client can decrypt its part of the ticket and thus knows what resource it may use. The client sends the ticket to the resource computer along with a fresh authenticator. (During initial logon, the resource computer is the client computer, and the service ticket is used locally.)
9. The resource computer (the client) validates the timestamp by checking whether the time is within the valid period, and then decrypts its portion of the ticket. This tells the computer which resource is requested and provides proof that the client has been authenticated. (Only the KDC would have a copy of the computer's password, and the KDC would not issue a ticket unless the client was authenticated.) The resource computer (the client) then uses an authorization process to determine whether the user is allowed to access the resource.

In addition to the authenticator and the use of computer passwords to encrypt ticket data, other Kerberos controls can be used. Tickets can be reused, but they are given an expiration date. Expired tickets can possibly be renewed, but the number of renewals can also be controlled.

In most implementations, however, Kerberos relies on passwords, so all the normal precautions about password-based authentication systems apply. If the user's password can be obtained, it makes no difference how strong the authentication system is—the account is compromised. However, there are no known successful attacks against Kerberos data available on the network. Attacks must be mounted against the password database, or passwords must be gained in an out-of-bounds attack (social engineering, accidental discovery, and so on).

One-Time Password Systems

Two problems plague passwords. First, they are (in most cases) created by people. Thus, people need to be taught how to construct strong passwords, and most people aren't taught (or don't care enough to follow what they're taught). These strong passwords must also be remembered and not written down, which means, in most cases, that long passwords cannot be required. Second, passwords do become known by people other than the individual they belong to. People do write passwords down and often leave them where others can find them. People commonly share passwords despite all your warnings and threats.

Passwords are subject to a number of different attacks. They can be captured and cracked, or used in a replay attack in which the passwords are intercepted and later used to repeat authentication.

One solution to this type of attack is to use an algorithm that requires the password to be different every time it is used. In systems other than computers, this has been accomplished with the use of a *one-time pad*. When two people need to send encrypted messages, if they each have a copy of the one-time pad, each can use the day's password, or some other method for determining which password to use. The advantage, of course, to such a system is that even if a key is cracked or deduced, it is only good for the current message. The next message uses a different key.

How, then, can this be accomplished in a computer system? Two current methods that use one-time passwords are time-based keys and sequential keys.

Time-Based Keys Time-based keys use hardware- or software-based authenticators that generate a random seed based on the current time of day. Authenticators are either hardware tokens (such as a key fob, card, or pinpad) or software. The authenticators generate a simple one-time authentication code that changes every 60 seconds. The user combines their personal identification number (PIN) and this code to create the password. A central server can validate this password, since its clock is synchronized with the token and it knows the user's PIN. Since the authentication code changes every 60 seconds, the password will change each time it's used.

This system is a two-factor system since it combines the use of something you know, the PIN, and something you have, the authenticator.

Sequential Keys Sequential key systems use a passphrase to generate one-time passwords. The original passphrase, and the number representing how many passwords will be generated from it, is entered into a server. The server generates a new password each time an authentication request is made. Client software that acts as a one-time generator is used on a workstation to generate the same password when the user enters the passphrase. Since both systems know the passphrase, and both systems are set to the same number of times the passphrase can be used, both systems can generate the same password independently.

The algorithm incorporates a series of hashes of the passphrase and a challenge. The first time it is used, the number of hashes equals the number of times the passphrase may be used. Each successive use reduces the number of hashes by one. Eventually, the number of times the passphrase may be used is exhausted, and either a new passphrase must be set or the old one must be reset.

When the client system issues an authentication request, the server issues a challenge. The server challenge is a hash algorithm identifier (which will be MD5 or SHA-1), a sequence number, and a seed (which is a cleartext character string of 1 to 16 characters). Thus, a server challenge might look like this: opt-md5 567 mydoghasfleas. The challenge is processed by the one-time generator and the passphrase entered by the user to produce a one-time password that is 64 bits in length. This password must be entered into the system; in some cases this is automatically done, in others it can be cut and pasted, and in still other implementations the user must type it in. The password is used to encrypt the challenge to create the response. The response is then returned to the server, and the server validates it.

The steps for this process are as follows (illustrated in Figure 7-2):

1. The user enters a passphrase.
2. The client issues an authentication request.
3. The server issues a challenge.

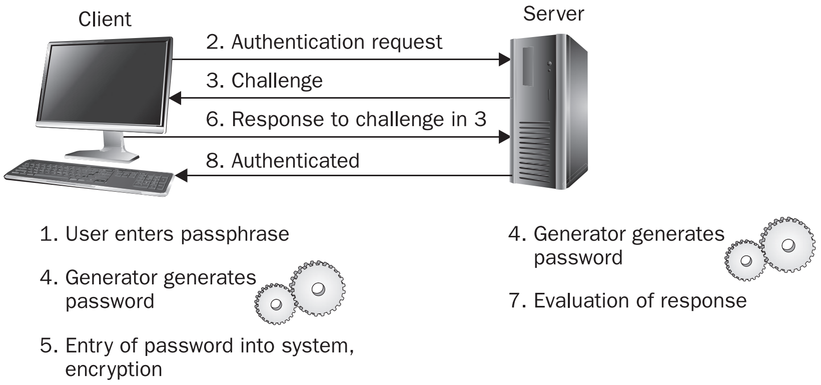
Client
Server
2. Authentication request
3. Challenge
6. Response to challenge in 3
8. Authenticated
1. User enters passphrase
4. Generator generates password
5. Entry of password into system, encryption
4. Generator generates password
7. Evaluation of response

Figure 7-2 The S/Key one-time password process is a modified challenge and response authentication system.

4. The generator on the client and the generator on the server generate the same one-time password.
5. The generated password is displayed to the user for entry or is directly entered by the system. The password is used to encrypt the response.
6. The response is sent to the server.
7. The server creates its own encryption of the challenge using its own generated password, which is the same as the client's. The response is evaluated.
8. If there is a match, the user is authenticated.

Sequential key systems, like other one-time password systems, do provide a defense against passive eavesdropping and replay attacks. There is, however, no privacy of transmitted data, nor any protection from session hijacking. A secure channel, such as IP Security (IPSec) or Secure Shell (SSH), can provide additional protection. Other weaknesses of such a system are in its possible implementations. Since the passphrase eventually must be reset, the implementation should provide for this to be done in a secure manner. If this is not the case, it may be possible for an attacker to capture the passphrase and thus prepare an attack on the system. Sequential key implementation in some systems leaves the traditional login in place. If users face the choice between using the traditional login and entering a complicated passphrase and then a long, generated password, users may opt to use the traditional login, thus weakening the authentication process.

Certificate-Based Authentication

A certificate is a collection of information that binds an *identity* (user, computer, service, or device) to the public key of a public/private key pair. The typical certificate includes information about the identity and specifies the purposes for which the certificate may be used, a serial number, and a location where more information about the authority that issued the certificate may be found. The certificate is digitally signed by the issuing authority, the certificate authority (CA). The infrastructure used to support certificates in an organization is called the Public Key Infrastructure (PKI). More information on PKI can be found in Chapter 10.

The certificate, in addition to being stored by the identity it belongs to, may itself be broadly available. It may be exchanged in e-mail, distributed as part of some application's initialization, or stored in a central database of some sort where those who need a copy can retrieve one. Each certificate's public key has its associated private key, which is kept secret, usually only stored locally by the identity. (Some implementations provide private key archiving, but often it is the security of the private key that provides the guarantee of identity.)

An important concept to understand is that unlike symmetric key algorithms, where a single key is used to both decrypt and encrypt, public/private key algorithms use two keys: one key is used to encrypt, the other to decrypt. If the public key encrypts, only the related private key can decrypt. If the private key encrypts, only the related public key can decrypt.

When certificates are used for authentication, the private key is used to encrypt or digitally sign some request or challenge. The related public key (available from the certificate) can be used by the server or a central authentication server to decrypt the request. If the result matches what is expected, then proof of identity is obtained. Since the related public key can successfully decrypt the challenge, and only the identity to which the private key belongs can have the private key that encrypted the challenge, the message must come from the identity. These authentication steps are as follows:

1. The client issues an authentication request.
2. A challenge is issued by the server.
3. The workstation uses its private key to encrypt the challenge.
4. The response is returned to the server.
5. Since the server has a copy of the certificate, it can use the public key to decrypt the response.
6. The result is compared to the challenge.
7. If there is a match, the client is authenticated.

Figure 7-3 illustrates this concept.

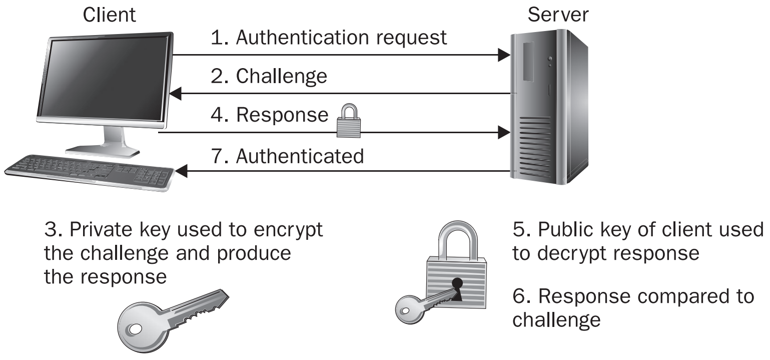
Client
Server
1. Authentication request
2. Challenge
4. Response
7. Authenticated
3. Private key used to encrypt the challenge and produce the response
5. Public key of client used to decrypt response
6. Response compared to challenge

Figure 7-3 Certificate authentication uses public and private keys.

It is useful here to understand that the original set of keys is generated by the client, and only the public key is sent to the CA. The CA generates the certificate and signs it using its private key, and then returns a copy of the certificate to the user and to its database. In some systems, another database also receives a copy of the certificate. It is the digital signing of the certificate that enables other systems to evaluate the certificate for its authenticity. If they can obtain a copy of the CA's certificate, they can verify the signature on the client certificate and thus be assured that the certificate is valid.

Two systems that use certificates for authentication are SSL/TLS and smart cards.

SSL/TLS

Secure Sockets Layer (SSL) is a certificate-based system that is used to provide authentication of secure web servers and clients and to share encryption keys between servers and clients. Transport Layer Security (TLS) is the Internet standard version (RFC 2246) of the proprietary SSL. While both TLS and SSL perform the same function, they are not compatible—a server that uses SSL cannot establish a secure session with a client that only uses TLS. Applications must be made SSL- or TLS-aware before one or the other system can be used.

NOTE While the most common implementation of SSL provides for secure communication and server authentication, client authentication may also be implemented. Clients must have their own certificate for this purpose, and the web server must be configured to require client authentication.

In the most commonly implemented use of SSL, an organization obtains a server SSL certificate from a public CA, such as VeriSign, and installs the certificate on its web server. The organization could produce its own certificate, from an in-house implementation of certificate services, but the advantage of a public CA certificate is that a copy of the CA's certificate is automatically a part of Internet browsers. Thus, the identity of the server can be proven by the client. The authentication process (illustrated in Figure 7-4) works like this:

1. The user enters the URL for the server in the browser.
2. The client request for the web page is sent to the server.
3. The server receives the request and sends its server certificate to the client.
4. The client's browser checks its certificate store for a certificate from the CA that issued the server certificate.
5. If the CA certificate is found, the browser validates the certificate by checking the signature on the server's certificate using the public key provided on the CA's certificate.
6. If this test is successful, the browser accepts the server certificate as valid.
7. A symmetric encryption key is generated and encrypted by the client, using the server's public key.
8. The encrypted key is returned to the server.
9. The server decrypts the key with the server's own private key. The two computers now share an encryption key that can be used to secure communications between the two of them.

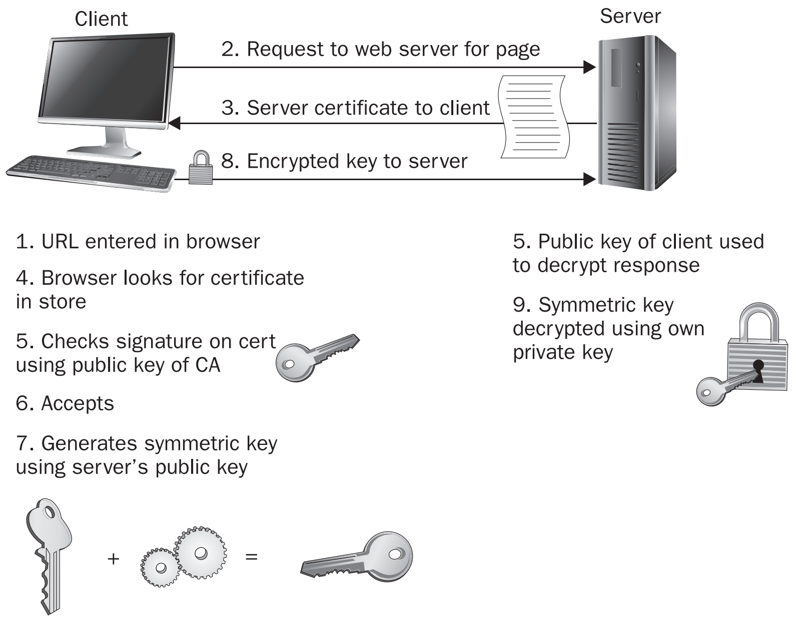
Client
Server
2. Request to web server for page
3. Server certificate to client
8. Encrypted key to server
1. URL entered in browser
4. Browser looks for certificate in store
5. Checks signature on cert using public key of CA
6. Accepts
7. Generates symmetric key using server's public key
5. Public key of client used to decrypt response
9. Symmetric key decrypted using own private key
+
=

Figure 7-4 SSL can be used for server authentication and to provide secure communications between a web server and a client.

There are many potential problems with this system:

- Unless the web server is properly configured to require the use of SSL, the server is not authenticated to the client, and normal, unprotected communication can occur. The security relies on the user using the https:// designation instead of http:// in their URL entry.
- If the client does not have a copy of the CA's certificate, the server will offer to provide one. While this ensures that encrypted communication between the client and the server will occur, it does not provide server authentication. The security of the communication relies on the user refusing to connect with a server that cannot be identified by a third party.
- The process for getting a CA certificate in the browser's store is not well controlled. In the past, it may have been a matter of paying a fee or depended on who you knew. Microsoft now requires that certificates included in its browser store are from CAs that pass an audit.
- Protection of the private key is paramount. While the default implementations only require that the key be in a protected area of the system, it is possible to implement hardware-based systems that require the private key to be stored only on a hardware device.

- As with any PKI-based system, the decision to provide a certificate to an organization for use on its web server is based on policies written by people, and a decision is made by people. Mistakes can be made. An SSL certificate that identifies a server as belonging to a company might be issued to someone who does not represent that company. And even though a certificate has expired, or another problem is discovered and a warning is issued, many users will just ignore the warning and continue on.

Smart Cards and Other Hardware-Based Devices

The protection of the private key is paramount in certificate-based authentication systems. If an attacker can obtain the private key, they can spoof the identity of the client and authenticate. Implementations of these systems do a good job of protecting the private key, but, ultimately, if the key is stored on the computer, there is potential for compromise.

A better system would be to require that the private key be protected and separate from the computer. Smart cards can be used for this purpose. While there are many types of smart cards, the ones used for authentication look like a credit card but contain a computer chip that is used to store the private key and a copy of the certificate, as well as to provide processing. Care should be taken to select the appropriate smart card for the application that will use them. Additional hardware tokens can be USB based and serve similar purposes. Smart cards require special smart card readers to provide communication between the smart cards and the computer system.

In a typical smart card implementation, the following steps are used to authenticate the client:

1. The user inserts the smart card into the reader (or moves it close to the scanner).
2. The computer-resident application responds by prompting the user for their unique PIN. (The length of the PIN varies according to the type of smart card.)
3. The user enters their PIN.
4. If the PIN is correct, the computer application can communicate with the smart card. The private key is used to encrypt some data. This data may be a challenge, or it may be the timestamp of the client computer. The encryption occurs on the smart card.
5. The encrypted data is transferred to the computer and possibly to a server on the network.
6. The public key (the certificate can be made available) is used to decrypt the data. Since only the possessor of the smart card has the private key, and because a valid PIN must be entered to start the process, successfully decrypting the data means the user is authenticated.

The use of smart cards to store the private key and certificate solves the problem of protecting the keys. However, user training must be provided so that users do not tape a written copy of their PIN to their smart card, or otherwise make it known. As in more traditional authentication systems, it is the person who possesses the smart card and PIN who will be identified as the user.

Smart cards are also extremely resistant to brute-force and dictionary attacks, since a small number of incorrect PIN entries will render the smart card useless for authentication. Additional security can be gained by requiring the presence of the smart card to maintain the session. The system can be locked when the smart card is removed. Users leaving for any amount of time can simply remove their card, and the system is locked against any individual who might be able to physically access the computer. Users can be encouraged to remove their cards by making it their employee ID badge and requiring the user to have their ID at all times. This also ensures that the smart card will not be left overnight in the user's desk. Yearly certificate updates can be required to help keep the security up to date.

Problems with smart cards are usually expressed in terms of management issues. Issuing smart cards, training users, justifying the costs, dealing with lost cards, and the like are all problems. In addition, the implementation should be checked to ensure that systems can be configured to require the use of a smart card. Some implementations allow the alternative use of a password, which weakens the system because an attack only needs to be mounted against the password—the additional security the smart card provides is eliminated by this ability to go around it. To determine whether a proposed system has this weakness, examine the documentation for this option, and look also for areas where the smart card cannot be used, such as for administrative commands or secondary logons.

Extensible Authentication Protocol (EAP)

The Extensible Authentication Protocol (EAP) was developed to allow pluggable modules to be incorporated in an overall authentication process. This means authentication interfaces and basic processes can all remain the same, while changes can be made to the acceptable credentials and the precise way that they are manipulated. Once EAP is implemented in a system, new algorithms for authentication can be added as they are developed, without requiring huge changes in the operating system. EAP is currently implemented in several remote access systems, including Remote Authentication Dial-In User Service (RADIUS).

Authentication modules used with EAP are called *EAP types*. Several EAP types exist, with the name indicating the type of authentication used:

- **EAP/TLS** Uses the TLS authentication protocol and provides the ability to use smart cards for remote authentication.
- **EAP/MD5-CHAP** Allows the use of passwords by organizations that require increased security for remote wireless 802.1x authentication but that do not have the PKI to support passwords.

Biometrics

In biometric methods of authentication, the "something you have" is something that is physically part of you. Biometric systems include the use of facial recognition and identification, retinal scans, iris scans, fingerprints, hand geometry, voice recognition, lip movement, and keystroke analysis. Biometric devices are commonly used today to provide authentication for access to computer systems and buildings, and even to permit pulling a trigger on a gun. In each case, the algorithm for comparison may differ, but a body part is examined and a number of unique points are mapped for comparison with stored mappings in a database. If the mappings match, the individual is authenticated.

The process hinges on two things: first, that the body part examined can be said to be unique, and second, that the system can be tuned to require enough information to establish a unique identity and not result in a false rejection, while not requiring so little information as to provide false positives. All of the biometrics currently in use have been established because they represent characteristics that are unique to individuals. The relative accuracy of each system is judged by the number of false rejections and false positives that it generates.

In addition to false negatives and false positives, biometrics live under the shadow, popularized by the entertainment industry, of malicious attackers cutting body parts from the real person and using them to authenticate to systems.

Other attacks on fingerprint systems have also been demonstrated—one such is the *gummy finger* attack. In May of 2002, Tsutomu Matsumoto, a graduate student of environment and information science at Yokohama National University, obtained an imprint of an audience member's finger and prepared a fake finger with the impression. He used about $10 of commonly available items to produce something the texture of the candy gummy worms. He then used the "gummy finger" to defeat ten different commercial fingerprint readers. While this attack would require access to the individual's finger, another similar attack was demonstrated in which Matsumoto used latent fingerprints from various surfaces. This attack was also successful. These attacks not only defeat systems most people believe to be undefeatable, but after the attack, you can eat the evidence!

Additional Uses for Authentication

Here are some additional uses for authentication:

- **Computer authenticating to a central server upon boot** In Windows, client computers are joined in the domain login at boot and receive security policy. Wireless networks may also require some computer credentials before the computer is allowed to have access to the network.
- **Computer establishing a secure channel for network communication** Examples of this are SSH and IPSec. More information on these two systems is included in the following sections.
- **Computer requesting access to resources** This may also trigger a request for authentication. More information can be found in the "Authorization" section later in this chapter.

SSH

Secure Shell (SSH) is available for most versions of Unix as well as for Windows systems. SSH provides a secure channel for use in remote administration. Traditional Unix tools do not require protected authentication, nor do they provide confidentiality, but SSH does.

IPSec

IP Security, commonly referred to as IPSec, is designed to provide a secure communication channel between two devices. Computers, routers, firewalls, and the like can establish IPSec sessions with other network devices. IPSec can provide confidentiality, data authentication, data integrity, and protection from replay. Multiple RFCs describe the standard.

Many implementations of IPSec exist, and it is widely deployed in virtual private networks (VPNs). It can also be used to secure communication on LANs or WANs between two computers. Since it operates between the network and transport layers in the network stack, applications do not have to be aware of IPSec. IPSec can also be used to simply block specific protocols, or communication from specific computers or IP address block ranges. When used between two devices, mutual authentication from device to device is required. Multiple encryption and authentication algorithms can be supported, as the protocol was designed to be flexible.

Authorization

The counterpart to authentication is *authorization*. Authentication establishes who the user is; authorization specifies what that user can do. Typically thought of as a way of establishing access to resources, such as files and printers, authorization also addresses the suite of privileges that a user may have on the system or on the network. In its ultimate use, authorization even specifies whether the user can access the system at all. There are a variety of types of authorization systems, including user rights, role-based authorization, access control lists, and rule-based authorization.

NOTE Authorization is most often described in terms of users accessing resources such as files or exercising privileges such as shutting down the system. However, authorization is also specific to particular areas of the system. For example, many operating systems are divided into user space and kernel space, and the ability of an executable to run in one space or the other is strictly controlled. To run within the kernel, the executable must be privileged, and this right is usually restricted to native operating system components.

User Rights

Privileges or *user rights* are different from permissions. User rights provide the authorization to do things that affect the entire system. The ability to create groups, assign users to groups, log in to a system, and many more user rights can be assigned. Other user rights are implicit and are rights that are granted to default groups—groups that are created by the operating system instead of by administrators. These rights cannot be removed.

In the typical implementation of a Unix system, implicit privileges are granted to the root account. This account is authorized to do anything on the system. Users, on the other hand, have limited rights, including the ability to log in, access certain files, and run applications they are authorized to execute.

On some Unix systems, system administrators can grant certain users the right to use specific commands as root, without issuing them the root password. An application that can do this, and which is in the public domain, is called sudo.

Role-Based Authorization (RBAC)

Each job within a company has a role to play. Each employee requires privileges (the right to do something) and permissions (the right to access particular resources and do specified things with them) if they are to do their job. Early designers of computer systems recognized that the needs of possible users of systems would vary, and that not all users should be given the right to administer the system.

Two early roles for computer systems were those of user and administrator. Early systems defined roles for these types of users to play and granted them access based on their membership in one of these two groups. Administrators (superusers, root, admins, and the like) were granted special privileges and allowed access to a larger array of computer resources than were ordinary users. Administrators, for example, could add users, assign passwords, access system files and programs, and reboot the machine. Ordinary users could log in and perhaps read data, modify it, and execute programs. This grouping was later extended to include the role of auditor (a user who can read system information and information about the activities of others on the system, but not modify system data or perform other administrator role functions).

As systems grew, the roles of users were made more granular. Users might be quantified by their security clearance, for example, and allowed access to specified data or allowed to run certain applications. Other distinctions might be made based on the user's role in a database or other application system. Commonly, roles are assigned by departments such as Finance, Human Resources, Information Technology, and Sales.

In the simplest examples of these role-based systems, users are added to groups that have specific rights and privileges. Other role-based systems use more complex systems of access control, including some that can only be implemented if the operating system is designed to manage them. In the Bell-LaPadula security model (described in Chapter 20), for example, data resources are divided into layers, or *zones*. Each zone represents a data classification, and data may not be moved from zone to zone without special authorization, and a user must be provided access to the zone to use the data. In that role, the user may not write to a zone lower in the hierarchy (from secret to confidential, for example), nor may they read data in a higher level than they have access to (a user granted access to the public zone, for example, may not read data in the confidential or secret zones).

The Unix role-based access control (RBAC) facility can be used to delegate administrative privileges to ordinary users. It works by defining role accounts, or accounts that can be used to perform certain administrative tasks. Role accounts are not accessible to normal logons—they can only be accessed with the `su` command (as described in Chapter 21).

Access Control Lists (ACLs)

Attendance at some social events is limited to invitees only. To ensure that only invited guests are welcomed to the party, a list of authorized individuals may be provided to those who permit the guests in. If you arrive, the name you provide is checked against this list, and entry is granted or denied. Authentication, in the form of a photo identification check, may or may not play a part here, but this is a good, simple example of the use of an access control list (ACL).

Information systems may also use ACLs to determine whether the requested service or resource is authorized. Access to files on a server is often controlled by information that is maintained on each file. Likewise, the ability for different types of communication to pass a network device can be controlled by ACLs.

File-Access Permissions

Both Windows and Unix systems use file permissions to manage access to files. The implementation varies, but it works well for both systems. It is only when you require interoperability that problems arise in ensuring that proper authorization is maintained across platforms.

Windows File-Access Permissions The Windows NTFS file system maintains an ACL for each file and folder. The ACL is composed of a list of access control entries (ACEs). Each ACE includes a security identifier (SID) and the permission(s) granted to that SID. Permissions may be either *access* or *deny*, and SIDs may represent user accounts, computer accounts, or groups. ACEs may be assigned by administrators, owners of the file, or users with the permission to apply permissions.

Part of the login process is the determination of the privileges and group memberships for the specific user or computer. A list is composed that includes the user's SID, the SIDs of the groups of which the user is a member, and the privileges the user has. When a connection to a computer is made, an access token is created for the user and attached to any running processes the user may start on that system.

Permissions in Windows systems are very granular. The permissions listed in Table 7-1 actually represent sets of permissions, but the permissions can be individually assigned as well.

When an attempt to access a resource is made, the security subsystem compares the list of ACEs on the resource to the list of SIDs and privileges in the access token. If there is a match, both of SID and access right requested, authorization is granted unless the access authorization is "deny." Permissions are cumulative (that is, if the read permission is granted to a user and the write permission is granted to a user, then the user has the read

Permission	If Granted on Folders	If Granted on Files
Full Control	All permissions	All permissions
Modify	List folder, read and modify permissions and attributes on the folder, delete the folder, add files to the folder	Read, execute, change, and delete files and file attributes
Read and Execute	List folder contents, read information on the folder including permissions and attributes	Read and execute the file; read information on the file, including permissions and attributes
List Folder Contents	Traverse folder (look and see folders within it), execute files in the folder, read attributes, list folders in the folder, read data, list the files within the folder	N/A
Read	List folder, read attributes, read permissions	Read the file, read attributes
Write	Create files, create folders, write attributes, write permissions	Write data to the file, append data to the file, write permissions and attributes
Special Permissions*	A granular selection of permissions	A granular selection of permissions

*These permissions do not match the permission groupings indicated. Each permission listed in the table can be applied separately.

Table 7-1 Windows File Permissions

and write permission), but the presence of a deny authorization will result in denial, even in the case of an access permission. The lack of any match results in an implicit denial.

It should be noted that file permissions and other object-based permissions in Windows can also be supplemented by permissions on shared folders. That is, if a folder is directly accessible from the network because of the Server Message Block (SMB) protocol, permissions can be set on the folder to control access. These permissions are evaluated along with the underlying permissions set directly on the folder using the NTFS permission set. In the case where there is a conflict between the two sets of permissions, the most restrictive permission wins. For example, if the share permission gives Read and Write permission to the Accountants group, of which Alice is a member, but the underlying folder permission denies Alice access, then Alice will be denied access to the folder.

Unix File-Access Permissions Traditional Unix file systems do not use ACLs. Instead, files are protected by limiting access by user account and group. If you want to grant read access to a single individual in addition to the owner, for example, you cannot do so. If you want to grant read access to one group and write access to another, you cannot. This lack of granularity is countered in some Unix systems (such as Solaris) by providing ACLs, but before we look at that system, we'll examine the traditional file protection system.

Information about a file, with the exception of the filename, is included in the *inode*. The file inode contains information about the file, including the user ID of the file's owner, the group to which the file belongs, and the *file mode*, which is the set of read/write/execute permissions.

File permissions are assigned to control access, and they consist of three levels of access: Owner, Group, and all others. Owner privileges include the right to determine who can access the file and read it, write to it, or, if it is an executable, execute it. There is little granularity to these permissions. Directories can also have permissions assigned to Owner, Group, and all others. Table 7-2 lists and explains the permissions.

ACLs are offered in addition to the traditional Unix file protection scheme. ACEs can be defined on a file and set through commands. These commands include information on the type of entry (the user or the ACL mask), the user ID (UID), group ID (GID), and the *perms* (permissions). The mask entry specifies the maximum permissions allowed for users (not including the owner) and groups. Even if an explicit permission has been granted for write or execute permission, if an ACL mask is set to read, read will be the only permission granted.

Permission	File users may…	Directory user may…
Read	Open and read contents of the file	List files in the directory
Write	Write to the file and modify, delete, or add to its contents	Add or remove files or links in the directory
Execute	Execute the program	Open or execute files in the directory; make the directory and the directories beneath it current
Denied	Do nothing	Do nothing

Table 7-2 Traditional Unix File Permissions

ACLs for Network Devices

ACLs are used by network devices to control access to networks and to control the type of access granted. Specifically, routers and firewalls may have lists of access controls that specify which ports on which computers can be accessed by incoming communications, or which types of traffic can be accepted by the device and routed to an alternative network. Additional information on ACLs used by network devices can be found in Chapters 13 and 14.

Rule-Based Authorization

Rule-based authorization requires the development of rules that stipulate what a specific user can do on a system. These rules might provide information such as "User Alice can access resource *Z* but cannot access resource *D*." More complex rules specify combinations, such as "User Bob can read file *P* only if he is sitting at the console in the data center." In a small system, rule-based authorization may not be too difficult to maintain, but in larger systems and networks, it is excruciatingly tedious and difficult to administer.

Compliance with Standards

If you are following a specific security framework, here's how NIST, ISO 27002, and COBIT tie in to this chapter.

NIST

NIST offers several publications relating to authentication:

- SP 800-120: Recommendation for EAP Methods Used in Wireless Network Access Authentication
- SP 800-63: Electronic Authentication Guideline
- SP 800-38B: Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication
- SP 800-38C: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality
- SP 800-25: Federal Agency Use of Public Key Technology for Digital Signatures and Authentication

ISO 27002

ISO 27002 contains the following provisions, to which this chapter's contents are relevant:

- **11.2.1** Formal user registration and deregistration procedures are used for granting access to information systems and services.
- **11.2.2** Allocation and use of privileges in information systems is restricted and controlled and allocated on a need-to-use basis only after formal authorization process.

- **11.2.3** Allocation and reallocation of passwords is controlled through a formal management process.
- **11.2.4** A process is used to review user access rights at regular intervals.
- **11.3.1** Guidelines are in place to advise users on selecting and maintaining secure passwords.
- **11.4.2** Authentication mechanisms are used for challenging external connections such as cryptography-based techniques, hardware tokens, software tokens, and challenge/response protocols.
- **11.5.2** Unique identifiers are provided to every user and the authentication method substantiates the claimed identity of the user.
- **11.5.3** A password management system enforces password controls, including enforcement of password changes, storage of passwords in encrypted form, and avoidance of onscreen display of passwords.

COBIT

COBIT contains the following provisions, to which this chapter's contents are relevant.

DS5.3:

- Ensure that all users and their activity on IT systems are uniquely identifiable.
- Enable user identities via authentication mechanisms.
- Confirm that user access rights to systems and data are in line with defined and documented business needs and that job requirements are attached to user identities.
- Ensure that user access rights are requested by user management, approved by system owners, and implemented by the security-responsible person.
- Maintain user identities and access rights in a central repository.
- Deploy cost-effective technical and procedural measures, and keep them current to establish user identification, implement authentication, and enforce access rights.

Summary

Authentication is the process of proving you are who you say you are. You can take that literally. If someone possesses your user credentials, it is possible for that person to say they are you, and to prove it to the satisfaction of the system. While many modern systems are based on hardware, such as tokens and smart cards, and on processes that can be assumed to be more secure, such as one-time passwords, most systems still rely on passwords for authentication. These systems are not well protected, because passwords are a terrible way to identify people. Other authentication methods are better. You should always evaluate an authentication system based on how easy it would be to defeat its controls.

Authorization, on the other hand, determines what an authenticated user can do on the system or network. A number of controls exist that can help define these rights of access explicitly. User rights are often provided directly by the operating system, either via permissions granted to the user account directly, or through the use of groups. If the user account belongs to a particular group, it is granted rights to do certain things. This method of authorization, while commonly found in most organizations, is not easy to manage and has a high potential for error. Role-based access controls are similar to group authorization, but they are organized into sets of functions based on some key common characteristic. Access control lists, in which specific, granular capabilities are individually specified, are also used to authorize functions.

References

Ballad, Bill, Tricia Ballad, and Erin Banks. *Access Control, Authentication, and Public Key Infrastructure.* Jones & Bartlett, 2010.

Clarke, Nathan. *Transparent User Authentication: Biometrics, RFID and Behavioural Profiling.* Springer, 2011.

Jain, Anil, Patrick Flynn, and Arun Ross, eds. *Handbook of Biometrics.* Springer, 2010.

Newman, Robert. *Security and Access Control Using Biometric Technologies.* Course Technology, 2009.

Nickell, Joe. *Real or Fake: Studies in Authentication.* The University Press of Kentucky, 2009.

Smith, Richard. *Authentication: From Passwords to Public Keys.* Addison-Wesley, 2001.

Todorov, Dobromir. *Mechanics of User Identification and Authentication: Fundamentals of Identity Management.* Auerbach Publications, 2007.